Amendments to the Specification:

Please replace the paragraph beginning at page 2, line 6, with the following rewritten paragraph:

However, the conversion of databases, including SQL databases, to such a model has required extensive manual input which is time-consuming and likely to result in errors. For example, auxiliary files must be produced to deploy the EJB's and referential integrity of the original database must be ensured. Moreover, these requirements and coordination between the database and the JAVA code can only be ~~validatedby~~ validated by a lengthy process of rigorous testing. The present state of the art does not support a simple, reliable, rapid method for performing the conversions, particularly in the volume that may be desired.

Please replace the paragraph beginning at page 7, line 11, with the following rewritten paragraph:

As an overview of the invention, as shown in Figure 1A, when it is desired to convert database files, the user invokes the Bean Grinder 10. A dialog with the user is then conducted to assemble necessary information in a plurality of steps within the bracket in Figure 1A labeled as an interface including steps 20 - 50. Specifically, the program presents the user with databases from which a selection 20 can be made. The user selects the databases to be converted. Then the user is presented with tables from which to select 30. After selecting the tables, the user specifies (40) the target location, that is, where the EJB's will be stored. The user then specifies (50) the application server on which the application will be run. Then the user begins the process of conversion by running the BeanGrinder 60, an overview of which is shown in Figure 1AA. Then, once the BeanGrinder 60 has been run, the program runs the

batch command file to generate the EJB file 70. Last, the program runs another batch command file to deploy the EJB Jar (Java ARchive) file in the application server 80, after which the program exits.

Please replace the paragraph beginning at page 9, line 1, with the following rewritten paragraph:

This function of producing auxiliary files is divided into ~~thre~~ three basic sections as shown in Figure 1C. Specifically, under control of a portion 62 of Bean Grinder 60, a descriptor file 63 (DescriptorFile) generates an XML deployment descriptor, a batch file 64 (JarBatFile) generates batch script to compile JAVA into CLASS files and compress the CLASS files into a JAR file and an an AGCommand 65 generates commands to import and deploy the EJB JAR file into a particular application server. AGCommand is one of two classes which produce data to assist in integrating the EJB's into a particular application server. The other is AGDeploy which generates a supplemental XML deployment file for a specific application server.

Please replace the paragraph beginning at page 10, line 1, with the following rewritten paragraph:

Figure 1H details the communications between user terminal 150, 150', host processor 100 and SQL databases ~~200~~ 160. The Bean Grinder is resident in processor 100 and is invoked 10 by normal communications between keyboad 150', processor 100 and display 150 and need not be further described or illustrated. Upon invocation 10 of the Bean Grinder, processor 100 accesses and communicates with SQL databases 160 as needed, placing retrieved information into the screens of Figures 1D - 1G which are displayed for user input. For example, upon invoking the

Bean Grinder, the first access of SQL databases will generally return the names or identifiers of existing databases and access information field queries which are presented to the user by means of the screen of Figure 1D or other prompt into which the information is inserted as a menu or the like. Then, the user selects, in sequence, the database(s) to be operated upon (20, I1000), including the database ID and password (I1010), the tables of the selected database (30, I1020), the target location 40 by directory path (I1030) and EJB Jar file name (I1040), and, finally the Java 2 Platform, Enterprise Edition (J2EE) application server (I1050). Once these selections are successfully made, the Bean Grinder processing may be initiated (1060), for example by pressing/clicking a button on the display. (Other such buttons preferably provided include a button for refreshing the table selection after switching databases and an exit button by which the process may be interrupted/aborted.) When processing is complete, an appropriate communication will be sent 1070 to the user terminal display. The application program then saves the information the user enters so the information can be used from session to session.

Please replace the paragraph beginning at page 12, line 1, with the following rewritten paragraph:

Referring now to Figure 2, the preferred embodiment of the data collection includes the following functions:

the application program querying 1105 the database 160 to get 1110 names of all tables related the database for a user to encapsulate the tables with an EJB;

and querying 1115 the database 160 to acquire 1120 information about fields within each table the user has selected, as shown in Figure 2. This provides the information concerning tables and fields in the selected database(s) as alluded to above to support construction of an EJBBeanFile object from each table and an attribute object from each field. Each EJBBeanFile has a collection of such attribute

objects which are assigned to it by the Bean Grinder. The EJBBeanFile also has methods to create JAVA source files for one EJB including a data file, an interface file which will be shared by the bean and the remote interface, a home interface file having "create" and "finder" methods, a Bean file which implements the home and remote interfaces, a Primary Key file which has a subset of attributes of the bean, a persistent file which extends the data file and has methods to create, read, update, and delete data and shields the client from having to access the home and remote interfaces, and a list file which is a collection of persistent objects. These types of files (and counterparts in other programming languages) will be familiar to those skilled in the art of object oriented computer programming. It should also be recognized that the development of these types of files is depicted ~~inf~~ in Figures 3 - 12.

Please replace the paragraph beginning at page 24, line 1, with the following rewritten paragraph:

writing a JAVA protected method 1898 named "getHome" to return an object of the Home class. (The body of this method provides a JAVA try-and-catch block to use an object of type InitialContext to look up the JNDI (Java Naming and Directory Interface) name. The JNDI name is preferably equal to the name of the selected table. These resulting JAVA constructs are stored, as desired, in memory 1805.